

Initials:

ID Number:

Do **not** write your full name on the test.

COMP103: Test

4 Sept, 2002.

Model Answers

Instructions

- Time: **2 hours**.
- Answer **all** the questions.
- There are 120 marks in total.
- Write your answers in the boxes in this test paper and hand in all sheets.
- Every box with a heavy outline requires an answer.
- If you think some question is unclear, ask for clarification.

Questions

Marks

1. Collection Types	[12]
2. "Big O" analysis	[12]
3. Implementing ArrayBag	[22]
4. Linked Lists	[25]
5. Using Vectors	[16]
6. Hash Tables	[18]
7. Binary Search and Sorting	[15]

Note: This test was considered fairly difficult. Also, in 2002 we covered slightly different material from this year. However, the questions will still be good practice for the 2003 test.

Question 1. Collection Types

[12 marks]

(a) [4 marks] Which of the following methods must be provided by a class that implements the `Bag` interface from the `jds` library? Mark the “yes” box or the “no” box for each method.

<code>public int <u>size</u>();</code>	yes: <input checked="" type="checkbox"/>	no: <input type="checkbox"/>
<code>public void <u>addElement</u>(Object value);</code>	yes: <input checked="" type="checkbox"/>	no: <input type="checkbox"/>
<code>public Object <u>elementAt</u>(int i);</code>	yes: <input type="checkbox"/>	no: <input checked="" type="checkbox"/>
<code>public boolean <u>containsElement</u>(Object value);</code>	yes: <input checked="" type="checkbox"/>	no: <input type="checkbox"/>
<code>public void <u>setElementAt</u> (Object value, int index);</code>	yes: <input type="checkbox"/>	no: <input checked="" type="checkbox"/>
<code>public void <u>removeElement</u>(Object value);</code>	yes: <input checked="" type="checkbox"/>	no: <input type="checkbox"/>
<code>public void <u>set</u> (Object key, Object value);</code>	yes: <input type="checkbox"/>	no: <input checked="" type="checkbox"/>
<code>public Enumeration <u>elements</u>();</code>	yes: <input checked="" type="checkbox"/>	no: <input type="checkbox"/>

(b) [4 marks] Which of the following methods must be provided by a class that implements the `Indexed` interface from the `jds` library? Mark the “yes” box or the “no” box for each method.

<code>public int <u>size</u>();</code>	yes: <input checked="" type="checkbox"/>	no: <input type="checkbox"/>
<code>public void <u>addElement</u>(Object value);</code>	yes: <input type="checkbox"/>	no: <input checked="" type="checkbox"/>
<code>public Object <u>elementAt</u>(int i);</code>	yes: <input checked="" type="checkbox"/>	no: <input type="checkbox"/>
<code>public boolean <u>containsElement</u>(Object value);</code>	yes: <input type="checkbox"/>	no: <input checked="" type="checkbox"/>
<code>public void <u>setElementAt</u> (Object value, int index);</code>	yes: <input checked="" type="checkbox"/>	no: <input type="checkbox"/>
<code>public void <u>removeElement</u>(Object value);</code>	yes: <input type="checkbox"/>	no: <input checked="" type="checkbox"/>
<code>public void <u>set</u> (Object key, Object value);</code>	yes: <input type="checkbox"/>	no: <input checked="" type="checkbox"/>
<code>public Enumeration <u>elements</u>();</code>	yes: <input checked="" type="checkbox"/>	no: <input type="checkbox"/>

(c) [4 marks] Consider the three collection types `Bag`, `Set`, and `Indexed`. For each type, state whether it allows duplicates or not and what constraints it places on the ordering of the items.

Bags and Sets are both collection of items where order does not matter, but Bags allow duplicates and Sets do not allow duplicates. Indexed collections allow duplicates, but the order of items does matter.

Question 2. Asymptotic or “Big O” analysis

[12 marks]

(a) [3 marks] What are the average case asymptotic costs (“Big O”) of the following sorting algorithms?

- Bubble Sort
- Insertion Sort
- Merge Sort
- Quicksort
- Selection Sort

$O(n^2)$

$O(n^2)$

$O(n \log(n))$

$O(n \log(n))$

$O(n^2)$

(b) [3 marks] What are the average case asymptotic costs of searching for an item in each of the following implementations of Bag? Assume that the size of the bag is n .

- ArrayBag (unordered, array)
- SortedVector (ordered, array)
- DoubleListBag (unordered, linked list)
- BucketHashtable (Hashtable, each bucket is an ArrayBag)
- QuadHashtable (Hashtable with open addressing, quadratic probing, and guaranteed less than 80% full).

$O(n)$

$O(\log(n))$

$O(n)$

$O(n)$

$O(1)$

(c) [3 marks] Suppose a program uses an algorithm with an average case asymptotic cost of $O(n^2)$. When the program is run on a case where $n = 1,000$, the measured running time of the program is 1.1 seconds. Give a reasonable estimate of the running time of the program on a case where $n = 4,000$.

$$4,000^2/1,000^2 \times 1.1 = 16 \times 1.1 = \text{about } 17 \text{ seconds}$$

(d) [3 marks] Explain why your estimate could be too high.

The $O(n^2)$ is only a limit for large enough n . The other factors (that are small for large enough n) have been dropped. However, these other factors might be significant for $n = 1000$. For example, it might be that the first 1 second is spent entirely on initialisation, and only the remaining 0.1 seconds depend on the size of the problem. That would mean that on the 4,000 case, the time would be more like $1 + 1.6 = 2.6$ seconds.

Question 3. Implementing ArrayBag

[22 marks]

The implementation of `ArrayBag` given in the lectures used an array to store the items in the collection. If the array is full, `addElement` creates a new array, double the size of the old one, and copies all the items over.

```
public void addElement (Object item) {
    if (elementCount == elementData.length){
        Object [ ] newArray = new Object[elementData.length*2];
        for (int i = 0; i < elementCount; i++)
            newArray[ i ] = elementData[ i ];
        elementData = newArray;
    }
    elementData[elementCount] = item;
    elementCount++;
}
```

(a) [4 marks] Explain why the average cost of adding an item to an `ArrayBag` containing n items is $O(1)$, even though it will take n steps whenever the array is full.

At each point where it has to double the array, it will take n steps to copy the items over into the new array. However, it can then add the next n items with just 1 step each. That means that averaged over the n steps, it has to do $2n$ steps to add n items, which means an average cost per item of just 2 steps, which is $O(1)$

(b) [4 marks] If an `ArrayBag` contains n items and `removeElement` has never been called, what is the largest possible amount of “wasted” space (array locations not currently used to hold items) that might be in the array?

($n-1$)

(c) [3 marks] The `removeElement` method does not reduce the size of the array when it removes items from the collection. If an `ArrayBag` contains n items, and `removeElement` has been called exactly m times, what is the largest possible amount of “wasted” space in the array?

The largest number of items the collection could have contained was $(n + m)$, at which point it might have had up to $(n + m - 1)$ wasted locations. Now, it has an extra m , so there might be up to $(n + 2m - 1)$ wasted locations.

(Question 3 continued on next page)

(Question 3 continued)

(d) [7 marks] The `ArrayBag` would use space more efficiently if `removeElement` checked the number of elements and replaced the array by an array of half the size whenever the array became only half full. Complete the following implementation of `removeElement` so that it does this.

```
public void removeElement (Object item) {
    for (int i = 0; i < elementCount; i++){
        if (item.equals(elementData[ i ])){
            elementCount--;
            elementData[i] = elementData[elementCount];

            if (elementCount <= elementData.length/2){
                Object [ ] newArray = new Object[elementData.length/2];
                for (int j = 0; j < elementCount; j++)
                    newArray[j] = elementData[j];
                elementData = newArray;
            }
            return;
        }
    }
    throw new NoSuchElementException();
}
```

(e) [2 marks] Explain why this particular design is a bad idea, by describing a sequence of `addElement` and `removeElement` operations that would be unnecessarily slow.

Add some number of elements (n) until the array has just been doubled. Then alternately remove an element and add an element, many times. Each remove and each add will cause the `ArrayBag` to first halve the array and then double the array, in order to remove and add just one element. That will mean $O(n)$ work on each step, just copying the array over, in addition to the cost of searching for the item to delete.

(f) [2 marks] Suggest a modification to your version of `removeElement` that would be more time efficient and still be reasonably space efficient.

Halve the array when it gets down to only one quarter full. Then, after halving the array, it can add lots of items before it has to double the array again.

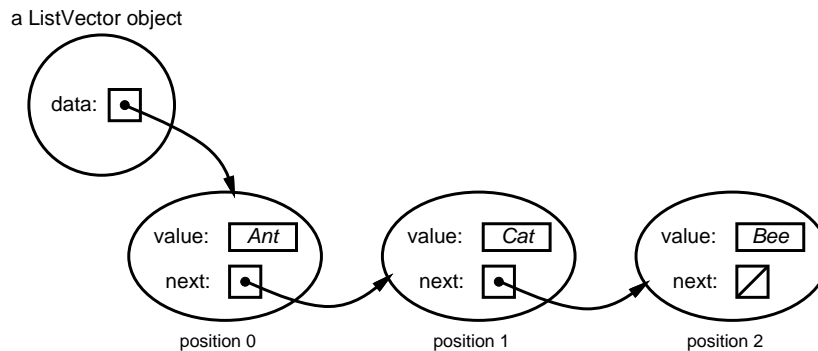
Question 4. Linked Lists

[25 marks]

The `Indexed` collection type describes collections of items in which the position of items in the collection is significant. The `Vector` class is an implementation of the `Indexed` type that uses an array to store the items. This question concerns the `ListVector` class that also implements the `Indexed` type, but uses a linked list data structure to store the items.

The `ListVector` class uses a singly linked list with no tail pointer. Part of the implementation is given below, along with a diagram of a `ListVector` value containing three items.

```
public class ListVector implements Indexed {
    private Node data;
    :
    private class Node{
        public Object value;
        public Node next;
        public Node(Object val, Node n){
            value = val;
            next = n;
        }
    }
}
```



You are to complete several of the method definitions for `ListVector` class.

(a) [4 marks] Complete the following definition of the `addFirst` method that inserts a new item into the collection at position 0 (the front of the list).

```
public void addFirst (Object val){
    data = new Node(val, data);
}
```

(b) [5 marks] Complete the following definition of the `size` method that returns the number of items in the collection.

(Question 4 continued on next page)

(Question 4 continued)

```
public int size() {  
    int count = 0;  
    for (Node rest = data; rest != null; rest = rest.next)  
        count++;  
    return count;  
}
```

(c) [8 marks] Complete the following definition of the `elementAt` method that returns the item at a given position in the collection. It throws a `NoSuchElementException` if there is no such position in the collection. Note that the first element in the collection is at position 0.

```
public Object elementAt (int index) {  
    if (index < 0 || data == null)  
        throw new NoSuchElementException(index);  
}
```

Question 5. Code with Vectors

[16 marks]

The following `countDoubles` method is supposed to find the number of items that occur exactly twice in a `Vector`. For example, when called on a vector containing the 11 items

“s”, “t”, “e”, “g”, “a”, “s”, “a”, “u”, “r”, “u”, “s”

`countDoubles` should return 2, because “a” and “u” occur exactly twice. (“s” occurs three times and should not be counted.) This version of `countDoubles` does not work correctly on all inputs.

```
public int countDoubles(Vector v){
    // Assumes that v does not contain any null items
    int numDoubles = 0;
    for (int i = 0; i < v.size(); i++){
        int count = 1;
        for (int j = i+1; j < v.size(); j++){
            if (v.elementAt(i).equals(v.elementAt(j)))
                count++;
        }
        if (count == 2)
            numDoubles++;
    }
    return numDoubles;
}
```

(a) [3 marks] Exactly how many times will the variable `count` be set to 1 if the vector `v` contains n items? (Express your answer in terms of n but do not use “Big O”.)

exactly n times

(b) [2 marks] What is the asymptotic cost (“Big O”) of the `countDoubles` method if the vector `v` contains n items?

$O(n^2)$. (The `equals` method, for example, will be called $n(n - 1)/2 = 1/2n^2 - 1/2n$ times, which is $O(n^2)$.)

(Question 5 continued on next page)

(Question 5 continued)

(c) [3 marks] Give an example of a vector for which `countDoubles` will return a *correct* value of 3.

["a", "a", "b", "b", "c", "c", "d", "e"]
 (any vector with three duplicates but no items 3 or more times)

(d) [3 marks] Give an example of a `Vector` for which `countDoubles` should return a value of 3, but actually returns an incorrect value. Also state what value `countDoubles` returns on your example.

["a", "a", "b", "b", "c", "c", "a", "e"],
 actual value = 4
 Any vector with at least one item that occurs 3 or more times.
`countDoubles` will actually return the number of distinct items that occur more than once, instead of the number of distinct items that occur exactly twice.

(e) [5 marks] Make minor corrections to the code for `countDoubles` so that it works correctly on all vectors.

[There are many ways of doing this. The problem is not counting the last two instances when the vector contains more than two copies of an item.

You could make a copy of the vector, and then remove items as you count them so you don't double count them.

You could sort the vector, and then you can count all the duplicates in a single pass (this is pretty efficient)

You could create a histogram using a `Map`, keeping a count in the `Map` of each item you have seen, and then scan the `Map` for items with a count of 2. (Using a hashtable implementation of a `Map`, this would be the most efficient method).

Two possible changes are given below. (Changes are underlined.)]

```

public int countDoubles(Vector v){
    int numDoubles = 0;
    for (int i = 0; i < v.size(); i++){
        int count = 0;
        for (int j = 0; j < v.size(); j++){
            if (v.elementAt(i).equals(v.elementAt(j)))
                count++;
        }
        if (count == 2)
            numDoubles++;
    }
    return numDoubles / 2;
}

```

```

public int countDoubles(Vector v){
    int numDoubles = 0;
    for (int i = 0; i < v.size(); i++){
        int count = 0;
        for (int j = i+1; j < v.size(); j++){
            if (v.elementAt(i).equals(v.elementAt(j)))
                count++;
        }
        if (count == 2)
            numDoubles++;
        else if (count == 3)
            numDoubles--;
    }
    return numDoubles;
}

```

Question 6. Hash Tables

[18 marks]

The `BucketHashtable` class from the `jds` library is implemented as an array of buckets, where each bucket is a `Bag` containing the items that hashed to that bucket. The buckets may be instances of any implementation of `Bag`.

The buckets could be instances of `ArrayBag`:

```
public class BucketHashtable implements Bag {
    private Bag[] buckets;
    public BucketHashtable(int numBuckets){
        buckets = new Bag[numBuckets];
        for (int i = 0; i < numBuckets; i++)
            buckets[ i ] = new ArrayBag();
    }
    :
```

(a) [4 marks] Suppose we have constructed a `BucketHashtable` with k buckets and have implemented the buckets with `ArrayBags`. If the `BucketHashtable` contains n items (where n is much larger than k), what is the average asymptotic (“Big O”) cost of searching for an item in the `BucketHashtable`? Justify your answer.

$O(n)$, since the size of each `ArrayBag` bucket will be n/k on average, and the cost of searching an `ArrayBag` is proportional to the number of items. Therefore, the cost of searching the hash table will be the cost of hashing ($O(1)$) plus a cost proportional to n/k , which is $O(n)$.

(b) [4 marks] On average, how much faster would it be to search for an item in the `BucketHashtable` than it would be to search for an item in an `ArrayBag` containing the same number of items? Justify your answer. (Assume that the total number of items is much greater than the number of buckets)

Roughly k times faster.
The cost of finding which bucket is small compared to searching down the items in each bucket. Since the cost of searching for an item in an `ArrayBag` is proportional to the size of the `ArrayBag`, and the size of a single `ArrayBag` containing all the items is, on average, k times the size of the `ArrayBag` in each bucket, searching a bucket should be approximately k times faster than searching a single `ArrayBag` with n .

(Question 6 continued on next page)

(Question 6 continued)

(c) [5 marks] Since the `BucketHashtable` class implements `Bag`, it would seem that you could use `BucketHashtables` for the buckets:

```
public class BucketHashtable implements Bag {
    private Bag[ ] buckets;
    public BucketHashtable(int numBuckets){
        buckets = new Bag[numBuckets];
        for (int i = 0; i < numBuckets; i++)
            buckets[ i ] = new BucketHashtable(numBuckets);
    }
    :
}
```

Explain why this would not work.

Each bucket would contain a `BucketHashtable` that contains an array of buckets, each of which is a `BucketHashtable`, which therefore contains an array of buckets, each of which is a `BucketHashtable`
The constructor of a `BucketHashtable` is a recursive procedure, and does not have a base case, so trying to construct a `BucketHashtable` would recurse until Java ran out of memory.

(d) [5 marks] The `QuadHashtable` class you wrote for assignment 5 also implements `Bag`. (`QuadHashtable` used open addressing, and quadratic probing.) `QuadHashtable` had the fastest search performance of any of the implementations of `Bag` we looked at. It would be possible to use `QuadHashtables` for the buckets in `BucketHashtable`:

```
public class BucketHashtable implements Bag {
    private Bag[ ] buckets;
    public BucketHashtable(int numBuckets){
        buckets = new Bag[numBuckets];
        for (int i = 0; i < numBuckets; i++)
            buckets[ i ] = new QuadHashtable(numBuckets);
    }
    :
}
```

Explain why this version of `BucketHashtable` could be even slower for searching than the original version using `ArrayBag` for the buckets.

The items that hash to a particular bucket would all hash to the same initial location in the `QuadHashtable` that was in that bucket (given that the tables are the same size, and assuming that the `QuadHashtable` uses the same hashing function as the `BucketHashtable`). Even with quadratic probing, all the items that hash to the same initial point end up in a run, so that the `QuadHashtable` would have to search along every item for every search. Since searching along the run involves squaring and adding integers, rather than just incrementing an integer, this would be slower than using an `ArrayBag`.
(This was a hard question!)

Question 7. Binary Search and Sorting Algorithms

[15 marks]

(a) [5 marks] Suppose a Vector contained the values

ant	bee	cat	dog	egg	fly	gnu	hen	jay	kit	man	owl	pig	rat	tui	yak
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Show the sequence of values from this vector that binary search would look at if it were searching for the value “egg”.

It depends a little on exactly which version of binary search you are using. It also depends whether you think that integer division rounds down (it does) or rounds up (it doesn't). Using the version given in the lectures and rounding down:

“hen”, “dog”, “fly”, “egg”

(b) [5 marks] Suppose a Vector containing the following values is to be sorted using Insertion Sort.

tui	bee	gnu	ant	pig	hen	yak	kit
0	1	2	3	4	5	6	7

Show the state of the vector at the end of the outer loop of Insertion Sort after each of the first three iterations:

after 1:	bee	tui	gnu	ant	pig	hen	yak	kit
	0	1	2	3	4	5	6	7
after 2:	bee	gnu	tui	ant	pig	hen	yak	kit
	0	1	2	3	4	5	6	7
after 3:	ant	bee	gnu	tui	pig	hen	yak	kit
	0	1	2	3	4	5	6	7

(Question 7 continued on next page)

(Question 7 continued)

(c) [5 marks] Suppose a Vector containing the following values is to be sorted using Merge Sort (recursive version).

tui	bee	gnu	ant	pig	hen	yak	kit
0	1	2	3	4	5	6	7

Show the state of the vector at the end of the call to Merge after each of the first three calls,

after 1:	bee	tui	gnu	ant	pig	hen	yak	kit
	0	1	2	3	4	5	6	7
after 2:	bee	tui	ant	gnu	pig	hen	yak	kit
	0	1	2	3	4	5	6	7
after 3:	ant	bee	gnu	tui	pig	hen	yak	kit
	0	1	2	3	4	5	6	7
